採用軟式 Tanner 圖於低密度奇偶檢查碼解碼

專題領域：通訊所

組　　別：A07

指導教授：翁永祿 副教授

主要聯絡人姓名：鄭弘得

主要聯絡人電話：0958266299

國立清華大學 電機工程學系

實作專題研究成果報告

LDPC decoding using soft Tanner graph
採用軟式 Tanner 圖於低密度奇偶檢查碼解碼

專題領域：通訊所

組　　別：A07

指導教授：翁詠祿 教授

組員姓名：黃昱霖、鄭弘得、江昱晟

研究期間：106 年 7 月 1 日至 107 年 6 月底止，計 12 個月

Abstract

With the development of machine learning and deep learning in recent years, applications have become more extensive, and even the field of communications has been greatly affected. In the process of communication, errors often occur due to noise interference. Therefore, in this topic, we combine machine learning and low-density parity check codes, and use machine learning to try to reduce decoding errors.

The method used in this topic is Belief propagation, also known as sum-product message passing. The algorithm we chose is the min-sum algorithm and use machine learning trains each signal's own weight. After comparing with the original method (fixed weight), it can reduce the error effectively after machine learning.

摘要

隨著近年來機器學習及深度學習的發展日益進步，應用也越加廣泛，就連通訊領域也深受其影響。在傳訊過程中，常常因為雜訊的干擾導致解碼產生誤差，因此在本專題中，我們結合機器學習與低密度奇偶檢查碼，並利用機器學習試圖降低解碼誤差。

本次我們專題使用的傳訊方法為置信度傳播（belief propagation），又稱為乘積和信息傳遞（sum-product message passing），而我們選擇的演算法為最小值-總和演算法(Min-Sum Algorithm)，並以機器學習 train 出每筆訊號專屬的 weight，經過與原先方法的比較(固定 weight)，經過機器學習過後確實能夠有效地降低誤差。

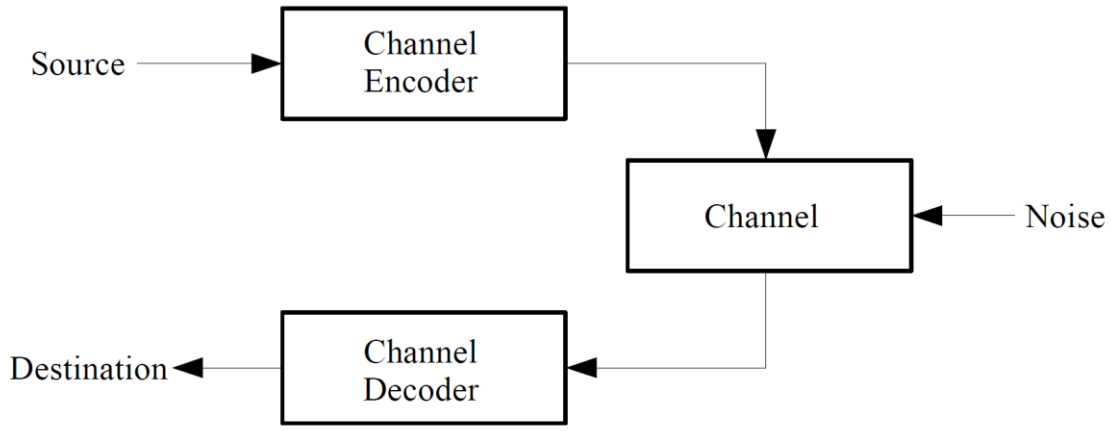
一、前言

低密度奇偶檢查碼（Low-density parity-check code，LDPC code）[2]，在1962 年被提出，並被證明其錯誤校正能力非常接近理論最大值。不過受限於當時技術，低密度奇偶檢查碼並無法實作。最近幾年，低密度奇偶檢查碼被重新發現，並隨著積體電路的技術演進，低密度奇偶檢查碼的實作逐漸可行，而成為各種先進通訊系統的頻道編碼標準。

研究目的：

在傳訊過程中，常常因為雜訊的干擾導致解碼產生誤差。在本專題中，除了透過普通的 LDPC code 來降低誤差外，更結合了機器學習。希望透過機器學習的方式[4]，有效的找出每筆訊號的 weight，並用以確實降低誤差。在參考論文[1]中，將此方法應用在低 SNR 的情況下，得到的結果相當不錯。而在本專題中，我們嘗試使用稍微高一點的 SNR，並檢視是否夠達成相同的效果。

背景簡介：



以上是一個通訊系統的基本架構圖，包含了各種不同階段的 encoder 與 decoder，我們這次主要在研究固定 weight 或使用各種不同 weight 的差別。使用機器學習找出適合各個訊號的 weight，並用於 decoder 使其降低誤差。

二、原理分析與系統設計

2.1 原理分析

(1) LDPC [2]

1. parity check matrix [2]

在介紹 LDPC 的 decoder 以及 encoder 之前，我們必須先了解所謂的 parity check matrix H ($H = [-A^T | I_{n-k}]$,)，由於 H 的定義是將矩陣右半部$(n-k)\times(n-k)$變成單位矩陣，因此我們必須將一開始得到的矩陣H'透過高斯消去法化簡成 H (也就是讓右半邊 n-k 的正方形便成單位矩陣)，再透過 H 計算後產生 G 來加密，而計算方法如下

$$H_{i\times j} = \left[-A^T_{i\times(j-i)} \middle| I_{ixi}\right] \Rightarrow G_{(j-i)\times j} = [I_{(j-i)\times(j-i)} | A_{(j-i)\times i}]$$

而 H 與 G 的關係為$G_{(j-i)\times j} \times H^T_{j\times i} = 0$，我們可以透過這個關係來驗證我們所計算出的 G 是否有錯誤。下圖為我們本次實驗中所使用的 H 矩陣

2. encoder

接下來是 encoder 的部分，我們將要傳遞的訊息 u 與 G 矩陣相乘得出一個加密過後的訊息 x

$$uG = [u_1, u_2, \dots, u_{(j-i)}] \begin{bmatrix} G_{11} & \cdots & G_{1j} \\ \vdots & \ddots & \vdots \\ G_{(j-i)1} & \cdots & G_{(j-i)j} \end{bmatrix} = x = [x_0, x_1, \dots, x_j]$$

原先訊息的長度為$(j-i)$bits，透過跟 G 相乘後，得到了一個 j bits 的加密訊息，而加密後的訊息 x 與 H 和 G 的關係為

$$xH^T = 0 \ (\because xH^T = uGH^T = u \cdot 0 = 0)$$

之後我們再將原本 010101 之訊息做轉換，將"0"轉為"+1"，"1"轉為"-1"以利後續 decoder 的運算。

3. decoder

以一個例子說明疊代的運作，如下 H 矩陣

$$H = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array} \begin{array}{c} \begin{array}{cccccccccc} V_1 & V_2 & V_3 & V_4 & V_5 & V_6 & V_7 & V_8 & V_9 & V_{10} \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

假設V_1~V_{10}的原始訊號分別為$(1.5, 1.8, -2, 3, -4, -2.5, -1, 0.5, 5, 3.4)$
下面以 Check Node C_1來看。

1. 找到最小值

在與C_1相連的 Variable Node 中，找出除了自己之外所有 Variable Node 絕對值之最小值。

2. 判斷正負號

在與C_1相連的 Variable Node 中，以除了自己之外所有 Variable Node 相乘來判斷正負號。若剩下數字相乘為正，則為加，若剩下數字相乘為負，則為減。

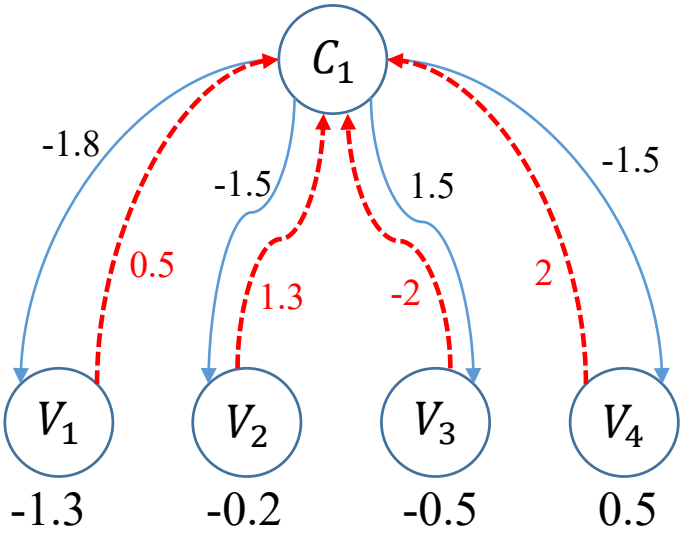
3. 更新 Variable Node

當所有的 Check Node 都做完上述兩個步驟之後，即可得出以下 table。將 Variable Node 的值加上所有在該 Variable Node 找到的值即完成 Variable Node 的更新。

原始訊號	1.5	1.8	-2	3	-4	-2.5	-1	0.5	5	3.4
	V_1	V_2	V_3	V_4	V_5	V_6	V_7	V_8	V_9	V_{10}
C_1	-1.8	-1.5	1.5	-1.5						
C_2	-1				1	1	1.5			
C_3		-0.5			0.5			-1.8	-0.5	
C_4				-1			3		-1	-1
SUM	-1.3	-0.2	-0.5	0.5	-2.5	-1.5	3.5	-1.3	3.5	2.4

4. 更新 Check Node

將更新的 Variable Node 值減去第一、二步驟所得出之值即可得出 Variable Node 回傳給 Check Node 之回傳值(如下圖紅色部分)。並以回傳值依照第一、二步驟做運算，即為新的 Check Node 值。



原始訊號	1.5	1.8	-2	3	-4	-2.5	-1	0.5	5	3.4
SUM	-1.3	-0.2	-0.5	0.5	-2.5	-1.5	3.5	-1.3	3.5	2.4
C_1回傳值	0.5	1.3	-2	2						
C_2回傳值	-0.3				-3.5	-2.5	2			
C_3回傳值		0.3			-3			0.5	4	
C_4回傳值				1.5			0.5		4.5	3.4
C_1'	-1.3	-0.5	0.5	-0.5						
C_2'	2				0.3	0.3	-0.3			
C_3'		-0.5			0.3			-0.3	-0.3	
C_4'				0.5			1.5		0.5	0.5

5. 重複上述步驟

重複疊代直到達成中止條件。

(2) Tanner graph

每個 parity-check matrix H 都有一個對應的 Tanner graph。Tanner graph 由三種元素構成：

－Variable Node：代表 H 中的每個 column。

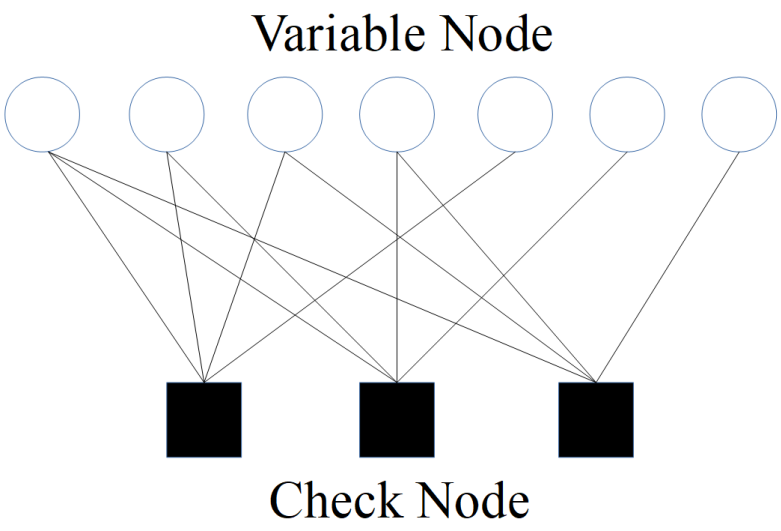
－Check Node：代表 H 中的每個 row。

－Edge：若$H_{mn} = 1$，則第 n 個 Variable Node 與第 m 個 Check Node 會連在一起，即為一條 Edge。

以一個例子說明 Tanner graph 的構造原理，如下 H 矩陣

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

而其 Tanner graph 如下圖



(3) Min-Sum Product Algorithm[3]

在介紹 Min-Sum Product Algorithm 之前，要先介紹一下 Sum product Algorithm。假設在一通訊系統的頻道有 AWGN 屬性，而傳送訊號為$u = [u_1, u_2, \dots]$，接收訊號是$y = [y_1, y_2, \dots]$，variable node 有 n 個，check node 有 m 個。而 Sum Product Algorithm 在解碼流程如下

Initialization :

Variable Node *n* Initialization :

$$\lambda_{n \to m}(u_n) = log \frac{P(u_n = 0/y_n)}{P(u_n = 1/y_n)} = \frac{2y_n}{\sigma^2}$$

Check Node m Initialization :

$$\Lambda_{m\to n}(u_n) = 0$$

Update :

Variable Node n Update :

$$\lambda_{n\to m}(u_n) = \frac{2y_n}{\sigma^2} + \sum_{m'\in M(n)\backslash m} \Lambda_{m'\to n}(u_n)$$

其中$m' \in M(n)\backslash m$是連接到 variable node n 的 check node 組合，但不包含第 m 個 check node

Check Node m Update :

$$\Lambda_{m\to n}(u_n) = 2tanh^{-1}\left\{\prod_{n'\in N(m)\backslash n} tanh\left[\frac{\lambda_{n'\to m}(u_{n'})}{2}\right]\right\}$$

其中$n' \in N(m)\backslash n$是連接到 check node m 的 variable node 組合，

但不包含第 n 個 bit node

Termination : 假設解碼後訊號為$\hat{u} = [\hat{u}_1, \hat{u}_2, \dots]$

$$\lambda_n(u_n) = \frac{2y_n}{\sigma^2} + \sum_{m'\in M(n)} \Lambda_{m\to n}(u_n)$$

$$\hat{u}_i = \begin{cases} 0, if\lambda_i \geq 0 \ \ for \ \ all \ \ i \in \{1,2,\dots,n\} \\ 1, if\lambda_i \leq 0 \ \ for \ \ all \ \ i \in \{1,2,\dots,n\} \end{cases}$$

而當$xH^T = 0$恆成立時，疊代終止(x 於上方提過)

當了解完 Sum Product Algorithm 後，我們只需改變 Check Node Update 的計算方式，便能得到 Min-Sum Product Algorithm。而改變計算式如下

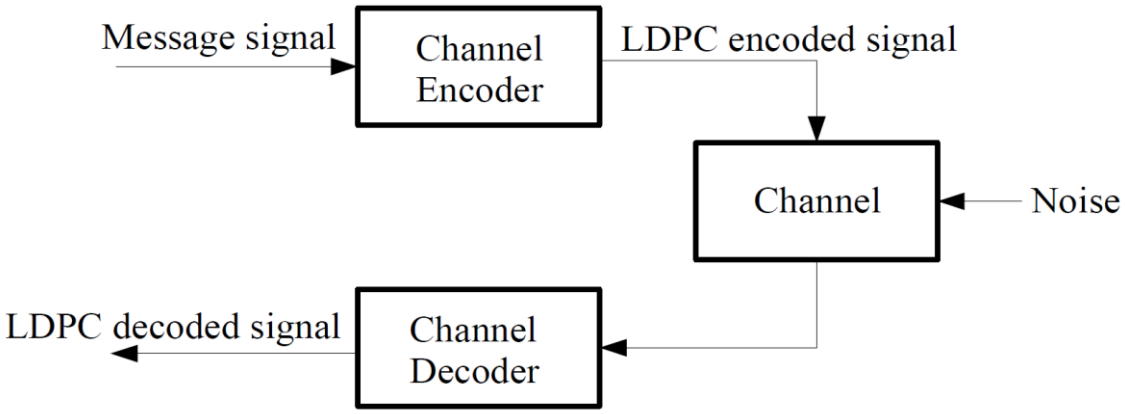
$$\sigma_m = XOR\{sgn(\lambda_{n'\to m}) | n' \in N(m)\backslash n\}$$

$$\mu_{m,min} = min\{|\lambda_{n'\to m}| | n' \in N(m)\backslash n\}$$

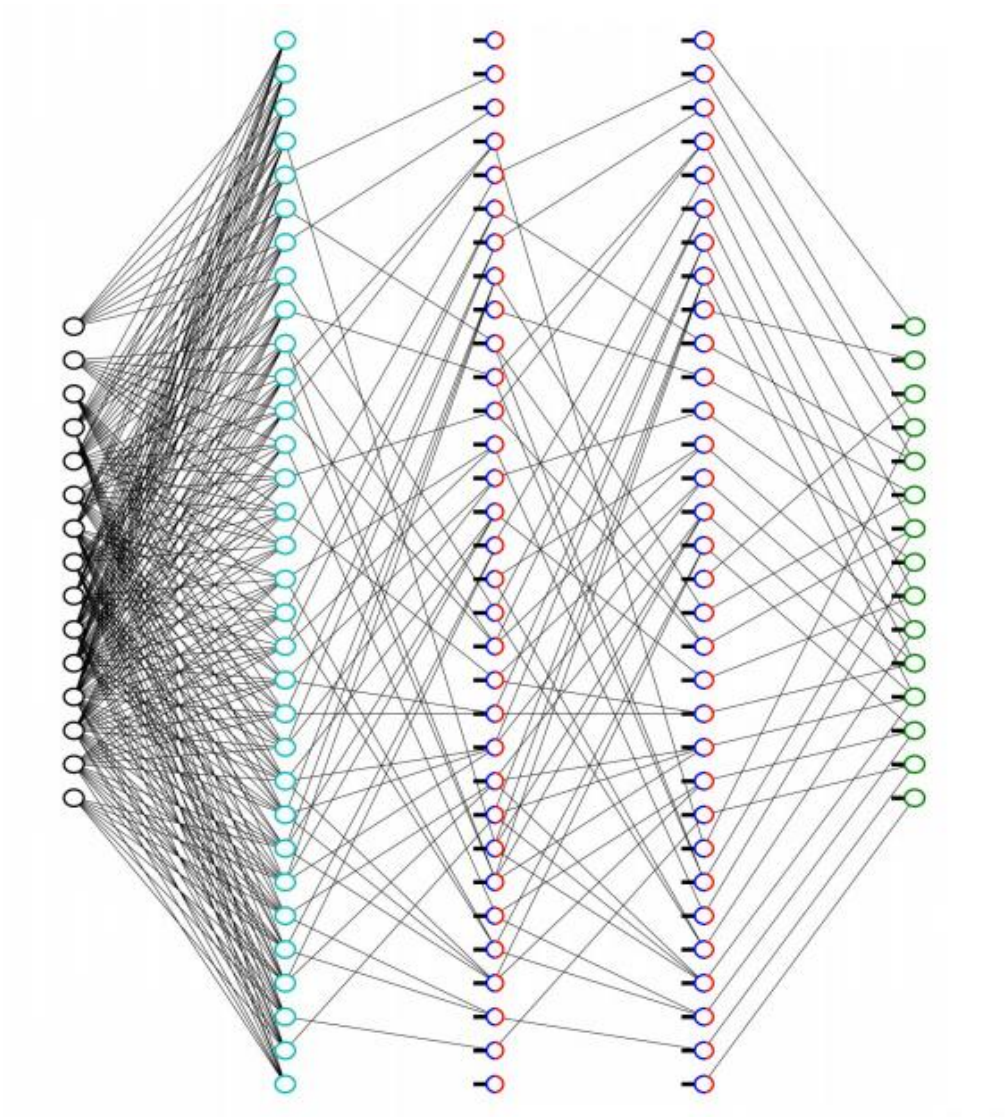
$$\Lambda_{m\to n}(u_n) = \sigma_m \cdot \mu_{m,min}$$

2.2 系統設計

(1)整體架構



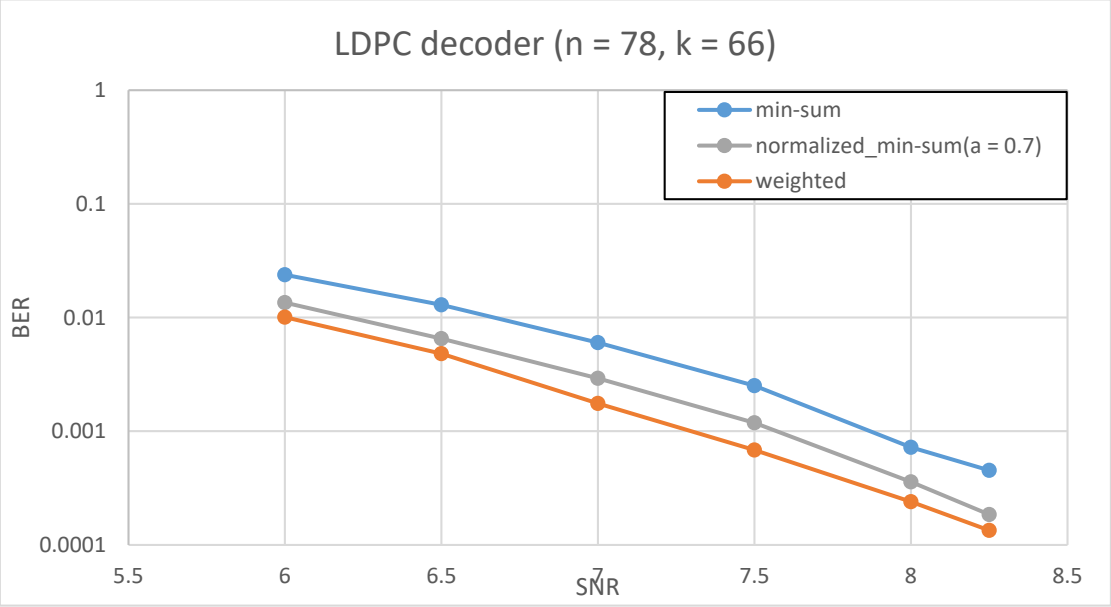
(2) deep learning model [1]



三、實驗結果

測試方式：在測試時每次產生66bit的message，首先經過encoder的編碼，加過程中加上SNR為6~8的noise，最後經過decoder的解碼。並比較signal跟message之差異，找出error bit。重複運行直到收集1000個error bit，並算出BER。

SNR	BER		
	min-sum	normalized min-sum	add_weight
6	0.023769697	0.022378788	0.010083333
6.5	0.012945455	0.011560606	0.00445
7	0.006015152	0.005242424	0.001757576
7.5	0.002515152	0.002106061	0.000683333
8	0.000721212	0.000651515	0.000239394
8.25	0.000451818	0.000184848	0.000133939



在上圖中，min-sum所使用的weight為固定的1，normalized min-sum所使用的weight為固定的0.8，而weighted所使用的為經過機器學習過後所找出的專屬weight。我們可以觀察到，在經過了機器學習過後，BER有明顯的降低。

四、結論

經過了機器學習所計算出的專屬weight確實可以使BER有明顯的降低，不僅是在參考論文中的低SNR，連在我們所使用的SNR中也可以得到不錯的結果。

五、參考文獻

[1] E. Nachmani, Y. Be'ery, and D. Burshtein, “Learning to decode linear codes using deep learning,” in *Proc. IEEE Annu. Allerton Conf. Commun. Control Comput. (Allerton)*, Monticello, IL, USA, 2016, pp. 341-346.

[2] R. G. Gallager, *Low Density Parity Check Codes*. Cambridge, Massachusetts: M.I.T. Press, 1963.

[3] M. P. C. Fossorier, M. Mihaljevic, and H. Imai, “Reduced complexity iterative decoding of low-density parity check codes based on belief propagation,” IEEE Trans. Commun., vol. 47, no. 5, pp. 673–680, May 1999.

[4] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016.